-------------------------Main class-----------------------  
  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.\*;  
import org.apache.hadoop.mapreduce.\*;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
public class MMDriver {  
  
    public static void main(String[] args) throws Exception {  
        if (args.length != 2) {  
            System.err.println("Usage: MatrixMultiply <in\_dir> <out\_dir>");  
            System.exit(2);  
        }  
        Configuration conf = new Configuration();  
        // M is an m-by-n matrix; N is an n-by-p matrix.  
        conf.set("m", "1000");  
        conf.set("n", "100");  
        conf.set("p", "1000");  
  
        Job job = new Job(conf, "MatrixMultiply");

        job.setJarByClass(MMDriver.class);  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(Text.class);  
  
        job.setMapperClass(MMMap.class);  
        job.setReducerClass(MMReduce.class);  
  
        job.setInputFormatClass(TextInputFormat.class);  
        job.setOutputFormatClass(TextOutputFormat.class);  
  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
        job.waitForCompletion(true);  
    }  
}

-------Mapper class------------------------  
  
import org.apache.hadoop.conf.\*;  
import [org.apache.hadoop.io](http://org.apache.hadoop.io/).LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
  
import java.io.IOException;  
  
public class MMMap  
  extends Mapper<LongWritable, Text, Text, Text> {  
        @Override  
        public void map(LongWritable key, Text value, Context context)  
                        throws IOException, InterruptedException {  
                Configuration conf = context.getConfiguration();  
                int m = Integer.parseInt(conf.get("m"));  
                int p = Integer.parseInt(conf.get("p"));  
                String line = value.toString();  
                // (M, i, j, Mij);  
                String[] indicesAndValue = line.split(",");  
                Text outputKey = new Text();  
                Text outputValue = new Text();  
                if (indicesAndValue[0].equals("M")) {  
                        for (int k = 0; k < p; k++) {  
                                outputKey.set(indicesAndValue[1] + "," + k);  
                                // outputKey.set(i,k);  
                                outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2]  
                                                + "," + indicesAndValue[3]);  
                                // outputValue.set(M,j,Mij);  
                                context.write(outputKey, outputValue);  
                        }  
                } else {  
                        // (N, j, k, Njk);  
                        for (int i = 0; i < m; i++) {  
                                outputKey.set(i + "," + indicesAndValue[2]);  
                                outputValue.set("N," + indicesAndValue[1] + ","  
                                                + indicesAndValue[3]);  
                                context.write(outputKey, outputValue);  
                        }  
                }  
        }  
}

-----------------------------Reducer---------------  
  
  
import org.apache.hadoop.io.Text;  
  
  
import java.io.IOException;  
import java.util.HashMap;  
  
public class MMReduce  
  extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> {  
        @Override  
        public void reduce(Text key, Iterable<Text> values, Context context)  
                        throws IOException, InterruptedException {  
                String[] value;  
                //key=(i,k),  
                //Values = [(M/N,j,V/W),..]  
                HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();  
                HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();  
                for (Text val : values) {  
                        value = val.toString().split(",");  
                        if (value[0].equals("M")) {  
                                hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));  
                        } else {  
                                hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));  
                        }  
                }  
                int n = Integer.parseInt(context.getConfiguration().get("n"));  
                float result = 0.0f;  
                float m\_ij;  
                float n\_jk;  
                for (int j = 0; j < n; j++) {  
                        m\_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;  
                        n\_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;  
                        result += m\_ij \* n\_jk;  
                }  
                if (result != 0.0f) {  
                        context.write(null,  
                                        new Text(key.toString() + "," + Float.toString(result)));  
                }  
        }  
}